



(12)

EUROPÄISCHE PATENTANMELDUNG

(21) Anmeldenummer: 90124244.6

(51) Int. Cl. 5 G06F 9/45

(22) Anmelddatum: 14.12.90

(30) Priorität: 15.12.89 US 451493

W-8000 München 2(DE)

(43) Veröffentlichungstag der Anmeldung:
19.06.91 Patentblatt 91/25

(72) Erfinder: Platoff, Michael, A.
2151 Ash Court

(44) Benannte Vertragsstaaten:
AT BE CH DE ES FR GB IT LI NL SE

Monmouth Junction, NJ 08852(US)
Erfinder: Wagner, Michael, E.

(71) Anmelder: Siemens Aktiengesellschaft
Wittelsbacherplatz 2

21 Crest Avenue

Trenton, NJ 08638(US)

(54) Verfahren für die automatische Syntaxanalyse (Parsen) des Textes von Computer-Programmen in Kompilierern.

(57) Zur Ermittlung und Darstellung der Struktur von Programmen, die wie in C Präprozessor-Anweisungen und Erweiterungen gegenüber der abstrakten Syntaxbaum-Darstellung aufweisen, wird ein modifizierter Parser-Generator mit zugehöriger Parser-Verwaltung verwendet, wobei mehrere Parser-Prozeduren denselben Eingabestrom vom Token verarbeiten und zusätzliche Operationen zum Abspalten neuer Parserprozeduren und zum Zusammenführen von aufgespaltenen Parser-Prozeduren sowie zum Austausch einer Parser-Prozedur für den einen Zweig

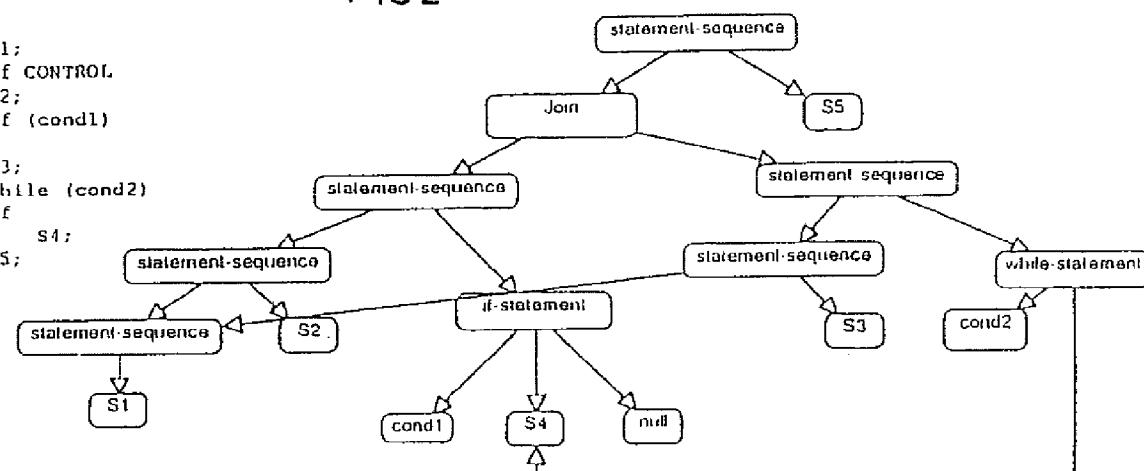
einer bedingten Kompilierungsanweisung durch die Parser-Prozeduren für den anderen Verzweigungspfad beim Auftreten von vorgegebenen bedingten Kompilierungsanweisungen ausgeführt werden. Welche der Parser-Prozeduren jeweils durch den Parsergenerator laufen soll, bestimmt die Parserverwaltung, wobei der jeweilige Status der einzelnen Parser-Prozeduren in der Parserverwaltung festgehalten wird.

FIG 2

```

S1;
1ifdef CONTROL
  S2;
  if (cond1)
  {
  }
  else
  {
    S3;
    while (cond2)
    {
    }
  }
  S4;
  S5;

```



VERFAHREN FÜR DIE AUTOMATISCHE SYNTAXANALYSE (PARSEN) DES TEXTES VON COMPUTER-PROGRAMMEN IN KOMPILIERERN

Die vorliegende Erfindung bezieht sich ganz allgemein auf die automatische Syntaxanalyse von Programmen: das Parsen und die Darstellung von Computer-Programmen.

Für die Umwandlung von Quellprogrammen in einer symbolischen Programmiersprache in entsprechende Zielprogramme einer gewünschten Maschinensprache werden Kompilierer verwendet, die zunächst das Quellprogramm analysieren und dann das entsprechende Zielprogramm erstellen. Im Rahmen der Quellprogrammanalyse ist u.a. eine Syntaxanalyse durchzuführen, um die Struktur des Quellprogramms richtig zu ermitteln und um das Quellprogramm strukturiert für die Weiterverarbeitung darzustellen, wozu sogenannte Parser verwendet werden, man siehe z.B. Jean Paul Tremblay und Paul G. Sorenson: "The Theory and Practice of Compiler Writing", Mc Graw - Hill Book Company, 1985.

Die vorliegende Erfindung ist vor allem dann sehr nützlich, wenn die Programmiersprache einen relativ lose integrierten Makro-Präprozessor aufweist. Obwohl die Erfindung nicht auf eine bestimmte Sprache beschränkt ist, empfiehlt sich für die vorliegende Erläuterung des Verfahrens die Verwendung der Programmiersprache C als Beispiel einer Sprache, die einen nur wenig integrierten Makro-Präprozessor besitzt - man siehe z.B. Computer Language (USA), Vol. 5, July 1988, Seiten 49 bis 54.

Ein bekannter Nachteil der Programmiersprache C ist die Tatsache, daß der Makro-Präprozessor nur sehr wenig mit der Grammatik von C zu tun hat. Dadurch ergeben sich eine Vielzahl an Problemen für Programmierwerkzeuge in C, zum Beispiel Struktur-Editoren, bildschirmorientierte Programm-Browser, oder Werkzeuge zur Programmumwandlung, in denen die Syntax und Semantik der Präprozessor-Anweisungen zusammen mit der zugrunde liegenden Programmstruktur berücksichtigt werden soll.

Programmdatenbanken, wie sie in integrierten Programmierumgebungen Verwendung finden, müssen alle Ausgangsinformationen eines Programmtexts enthalten, damit den Programmierern alle strukturellen Informationen zur Verfügung stehen. In vielen dieser Umgebungen werden Syntax und Semantik der Programme häufig in Form von abstrakten Syntaxbäumen dargestellt. Abstrakte Syntaxbäume sind leicht zu erzeugen und zu bearbeiten. Zudem erlauben sie eine brauchbare Wiedergabe der Programmstruktur. Für die Darstellung von Programmen in der Sprache C sind abstrakte Syntaxbäume jedoch nicht ausreichend. Der C-Prä-

prozessor (cpp) unterstützt durch die bedingte Kompilierung und die Makrosubstitution Eigenschaften, die mit einer Baumstruktur nicht beschrieben werden können und auf vorhandene Parser-Methoden nicht anwendbar sind. Zum Beispiel

stellt bereits ein Programm mit einer einzigen `#if`-Anweisung im Grunde genommen zwei Programme dar: ein Programm, in dem der Ausdruck der Bedingung zutrifft, und ein weiteres Programm, in dem der Ausdruck nicht zutrifft. Viele Informationen werden durch den Einsatz von cpp entfernt, auf die der Parser später nicht mehr zurückschliessen kann. Aber auch wenn der Parser auf diese Informationen zurückgreifen könnte, wäre es sehr schwierig, diese beiden Aspekte des Programms in einem abstrakten Syntaxbaum darzustellen. Es ist demnach ein "Wald" von Syntaxbäumen zur vollständigen Darstellung erforderlich: ein abstrakter Syntaxgraph. Bei den meisten C-Compilern und deren Hilfsprogrammen werden die Programme zuerst mit einem Makro-Präprozessor verarbeitet. Der daraus erzeugte Text wird anschließend in einer lexikalischen und syntaktischen Analyse weiterverarbeitet. In vieler Hinsicht hat der C-Präprozessor sehr viel dazu beigetragen, daß C als Systemprogrammiersprache so leistungsfähig ist. Die folgende kommentierte Aufstellung der Präprozessor-Anweisungen verdeutlicht die Stärken des Präprozessors und zeigt einige der Schwierigkeiten in der Anwendung auf:

`#define name token-string`

ersetzt nachfolgende Stellen von Name durch Token-String.

`#define name (arg [, arg] ...) token-string`

definiert ein parametrisiertes Makro. Makrodefinitionen bestehen aus einem Makroblock, auf die optional formale Parameter folgen können. Die Definition eines Makroblocks ist eine beliebige Folge von Token. Innerhalb des Makroblocks werden die Parameter bei der Verwendung des Makros an Ort und Stelle durch die im Aufruf des Makros angegebenen Argumente ersetzt.

`#include "file name"`

fügt den Inhalt einer Datei in eine andere Datei ein. Diese Anweisung wird meistens zur Übernahme der Schnittstellendefinitionen anderer Module verwendet.

`#if constant-expression`

schließt bestimmte Textteile einer Quelldatei wahlweise ein oder aus. Die `#if`-Anweisung weitet den Konstantenausdruck aus und übernimmt den Text, der vor der zugehörigen `#else`-, `#elif`- oder `#endif`-Anweisung steht, falls der Ausdruck zutrifft. Durch diese Möglichkeit lassen sich Mo-

diale erstellen, die über verschiedene Betriebssystemumgebungen und Rechnerarchitekturen portiert werden können. Diese Eigenschaft erlaubt außerdem die bedingte Übernahme verschiedener Eigenschaften in einem Modul.

#ifdef name

ist eine ähnliche Anweisung wie **#if**, mit dem Unterschied, daß der Text, der zwischen der nächsten **#else**-, **#elif**- oder **#endif**-Anweisung liegt, nur dann aufgenommen wird, wenn name definiert ist.

Die Integration des Präprozessors in die Phase der Syntaxanalyse wirft verschiedene Probleme auf. Zum Beispiel ist die Grammatik des Präprozessors nicht auf die Grammatik von C abgestimmt. Da die Präprozessor-Anweisungen an jeder beliebigen Stelle zwischen den Programm-Tokens von C stehen können, lassen sich die beiden Grammatiken nicht miteinander verknüpfen. Zudem führen Anweisungen zur bedingten Kompilierung zwangsläufig zu mehreren Parserbäumen oder Versionen für ein und dasselbe Modul. Idealerweise sollten die syntaktischen und semantischen Informationen dieser dadurch hervorgerufenen Versionen als integrierte Programmstruktur dargestellt werden. In einer integrierten Darstellung der Programmstruktur sollten Makros zum einen einfach als Makros, ähnlich der Syntax von Bezeichnern oder Funktionsaufrufen, zum anderen aber auch in ihren erweiterten Formen zugänglich sein. Eine weitere Schwierigkeit ergibt sich dadurch, daß der Inhalt einer Makrodefinition keine abgeschlossene syntaktische Einheit der zugrundeliegenden C-Grammatik sein muß. Eine syntaktische Einheit kann also auch auf mehrere Dateien verteilt sein.

Im Zusammenhang mit der vorliegenden Erfindung ermöglicht ein modifizierter Parser-Generator die Analyse von Programmen, die wie C-Programme Präprozessor-Anweisungen enthalten und Erweiterungen gegenüber der Darstellung in einem abstrakten Syntaxbaum umfassen, damit eine integrierte Darstellung der Programmstruktur möglich wird.

Im Rahmen der Erfindung wird der Parser-Generator dazu verwendet, einen Parser für Programmiersprachen zu erzeugen, die wie C Präprozessor-Anweisungen enthalten. Die Erweiterungen gegenüber der normalen Darstellung von Programmen mit einem Syntaxbaum werden außerdem dazu verwendet, eine integrierte Darstellung der Struktur anzulegen.

Gemäß der Erfindung umfaßt das Verfahren für die automatische Syntaxanalyse eines Computer-Programmtextes, in dem bedingte Kompilierungsanweisungen enthalten sind, durch die sich mehrere Versionen eines Programms aus einem einzigen Quelltext ergeben, folgende Verfahrensschritte:

- * Multiplexen und Synchronisieren mehrerer

gleichzeitig aktiver Parser-Prozeduren, die einen gemeinsamen Eingabestrom des Textes bearbeiten;

- 5 * Abspalten einer neuen Parser-Prozedur beim Auftreten von bedingten Kompilierungsanweisungen im Eingabestrom, die darauf hinweisen, daß direkt nachfolgende Eingaben eine Variante aus einer Vielfachheit von möglichen Programmvarianten darstellen, als ParseFork-Operation;
- 10 * Austausch einer Parser-Prozedur, die einem Verzweigungspfad der bedingten Kompilierungsanweisung zugeordnet ist, durch die Parser-Prozedur des anderen Verzweigungspfads der bedingten Kompilierungsanweisung als ParseExchange-Operation;
- 15 * Zusammenführen von zwei zuvor aufgespalteten Parser-Prozeduren, falls alle Kriterien für eine Zusammenführung erfüllt sind, als ParseJoin-Operation, sowie
- 20 * Festlegen der Reihenfolge der Shift-Reduktions-Operationen in den miteinander synchronisierten Parser-Prozeduren.

Weiterbildungen dieses Verfahrens bezüglich

- 25 der Steuerung der Abspaltung einer Parser-Prozedur, der Durchführung von Reduktionen, der Zusammenführung von aufgespalteten Parser-Prozeduren sowie des gesamten Verfahrensablaufes ergeben sich aus den weiteren Ansprüchen.

- 30 Die Funktionsweise der Erfindung wird nachfolgend anhand eines in der Zeichnung dargestellten Ausführungsbeispiels erläutert.

FIG 1 zeigt einen Ausschnitt aus einem C-Programm mit einer bedingten Kompilierung und dem entsprechenden abstrakten Syntaxbaum, der sich ergibt, wenn das Präprozessor-Symbol CONTROL definiert ist (linker Teil) und wenn es nicht definiert ist (rechter Teil), und

FIG 2 stellt den abstrakten Syntaxgraphen für das Beispiel in FIG 1 dar.

An der Zeichnung kann man erkennen, daß durch die Tatsache, daß sich durch bedingte Kompilierungsanweisungen mehrere Versionen aus einem Programmmodul ergeben, Methoden erforderlich sind, mit denen mehrere Versionen eines Programms von einem Parser analysiert und die einzelnen Versionen zusammen in einer Struktur dargestellt werden können.

Für die Ausführung der vorliegenden Erfindung, ist ein LALR-Modul besonders geeignet, und zwar wegen der Allgemeinheit des Shift-Reduktions-Parser-Modells, wegen der ausgereiften C-Tools, die mit Parser-Generatoren, wie YACC und Bison erstellt wurden, wegen der Anforderung, daß diese Tools die Präprozessor-Anweisungen verarbeiten, sowie wegen der Umgebung, in der sich der analy-

sierte Code befindet, der eine sauber getrennte Lösung ermöglichte.

Um die bedingte Kompilierung im aktuellen Modell zu verarbeiten, wird ein veränderter LALR-Parser-Treiber verwendet, damit die gleichzeitige synchron erfolgende Analyse eines einzigen Eingabestroms von Token durchgeführt werden kann. Das erweiterte Parser-Modell umfaßt die folgenden drei Operationen, die jeweils mit einer anwendungsspezifischen Rücksprung-Routine verbunden sind und zum Erstellen der abstrakten Syntaxgraphen verwendet werden.

ParseFork: spaltet eine neue Parser-Prozedur ab, wenn eine Anweisung wie `#ifdef`, `#ifndef` oder `#if` im Eingabestrom angetroffen wird. Während der Abspaltung wird der aktuelle Zustand des Parsers kopiert. Dazu zählen der Status-Stack und der Werte-Stack. Der Zeiger auf den Eingabestrom wird von allen Parser-Prozeduren gemeinsam verwendet.

ParseJoin: versucht eine Zusammenführung von zwei Parser-Prozeduren, wenn eine `#endif`-Anweisung im Eingabestrom angetroffen wird. Die beiden Verzweigungen einer Parser-Prozedur können nicht immer sofort zusammengeführt werden, da die Parser-Prozeduren unter Umständen die Kriterien für eine Zusammenführung nicht erfüllen. Deshalb wird die Zusammenführung der beiden Parser-Prozeduren so lange verschoben, bis beide Parser-Prozeduren alle Kriterien für die Zusammenführung erfüllen. Wenn Parser-Prozeduren für eine Zusammenführung anstehen, bleiben mehrere Parser-Prozeduren aktiv (lesen also auch weiterhin den Eingabestrom), bis die beiden Parser-Prozeduren zusammengeführt werden können. Die Rücksprung-Routine wird in diesem Fall erst zum Zeitpunkt der Zusammenführung aufgerufen. Die Verschiebung der Zusammenführung der Parser-Prozeduren ist eines der schwierigsten Probleme, die sich stellen, was nachfolgend ausführlich besprochen wird.

ParseExchange: tauscht zwei Verzweigungen einer aufgespaltenen Parser-Prozedur aus. Diese Operation wird vorgenommen, wenn eine Anweisung wie `#else` oder `#elif` im Eingabestrom als Token angetroffen wird. Die aktuell ausgeführte Parser-Prozedur wird ausgesetzt und die zu diesem Zeitpunkt ausgesetzten Parser-Prozeduren werden nach einer `#else`- oder `#elif`-Anweisung aktiviert und für die Aufnahme von Token vorbereitet.

Der Parser-Algorithmus ist eine dahingehende Abwandlung des standardmäßigen LALR-Algorithmus, einen einzigen Eingabestrom in mehreren Parser-Prozeduren verarbeiten zu können. Der nachstehend angegebene Algorithmus besteht aus zwei Prozeduren: einer Parser-Verwaltung und ei-

nem modifizierten LALR-Treiber.

Die Parser-Verwaltung multiplext den Eingabestrom und bestimmt, welche Parser-Prozedur durch den LALR-Treiber laufen soll. Der Status der einzelnen Parser-Prozeduren wird in einer Status-Variablen in der Parser-Verwaltung festgehalten. Der Status kann folgende Einstellungen haben:

5 **Active** gibt an, daß die Parser-Prozedur das aktuelle Token aus dem Eingabestrom entgegennehmen darf.

10 **BlockedOnInput** gibt an, daß das aktuelle Token von der Parser-Prozedur bereits erkannt wurde und keine weiteren Operationen durchgeführt werden können, da die anderen aktiven Parser-Prozeduren bereits darauf warten, das aktuelle Token zu verarbeiten.

15 **Sleeping** gibt an, daß eine Parser-Prozedur ausgesetzt wurde, da sie nichts mit dem aktuellen Eingabestrom zu tun hat. Diese Situation tritt dann ein, wenn eine Verzweigung einer bedingten Kompilierungsanweisung verarbeitet wird und die andere Verzweigung auf die entsprechende `#else`- oder `#endif`-Anweisung wartet.

20 Ein wesentlicher Bestandteil der Parser-Verwaltung und des modifizierten LALR-Treibers ist der Mechanismus und die Bedingungen für die Zusammenführung der aufgespaltenen Parser-Prozedur. An jeder Verzweigungsstelle wird ein Zeiger auf die Spitze des Status-Stacks mitgeführt. Dieser Zeiger zeigt auf den letzten Status, den die beiden Parser-Prozeduren vor der Aufspaltung gemein hatten. Wenn also an einer Stelle, an der noch die Verarbeitung einer `#endif`-Anweisung oder eine Zusammenführung ansteht, eine Reduktion durchgeführt wird, die dazu führt, daß der Status-Stack unter die Stelle mit der Verzweigung rutscht, muß der Zeiger angepaßt werden, um auf die nächste Stelle zu verweisen, an der die beiden Parser-Prozeduren wieder zusammengeführt werden könnten. Für diese Zeiger wird ein Stack verwendet, um Verschachtelungen von bedingten Kompilierungsanweisungen verarbeiten zu können.

25 Für eine geordnete Zusammenführung müssen eine Reihe von Kriterien erfüllt sein. Jede Parser-Prozedur hat einen Status-Stack und einen Werte-Stack, die von Aktionsroutinen verwendet werden und aufeinander abgestimmt sind. Zwei Parser-Prozeduren lassen sich dann sicher zusammenführen, wenn keine Informationen bei der Zusammenführung verlorengehen können. Das bedeutet, daß die beiden Parser-Prozeduren zusammengeführt werden können, wenn sie den gleichen Status-Stack besitzen und durch die Zusammenführung keine Werte auf dem Werte-Stack verlorengehen.

30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140 145 150 155 160 165 170 175 180 185 190 195 200 205 210 215 220 225 230 235 240 245 250 255 260 265 270 275 280 285 290 295 300 305 310 315 320 325 330 335 340 345 350 355 360 365 370 375 380 385 390 395 400 405 410 415 420 425 430 435 440 445 450 455 460 465 470 475 480 485 490 495 500 505 510 515 520 525 530 535 540 545 550 555 560 565 570 575 580 585 590 595 600 605 610 615 620 625 630 635 640 645 650 655 660 665 670 675 680 685 690 695 700 705 710 715 720 725 730 735 740 745 750 755 760 765 770 775 780 785 790 795 800 805 810 815 820 825 830 835 840 845 850 855 860 865 870 875 880 885 890 895 900 905 910 915 920 925 930 935 940 945 950 955 960 965 970 975 980 985 990 995 1000 1005 1010 1015 1020 1025 1030 1035 1040 1045 1050 1055 1060 1065 1070 1075 1080 1085 1090 1095 1100 1105 1110 1115 1120 1125 1130 1135 1140 1145 1150 1155 1160 1165 1170 1175 1180 1185 1190 1195 1200 1205 1210 1215 1220 1225 1230 1235 1240 1245 1250 1255 1260 1265 1270 1275 1280 1285 1290 1295 1300 1305 1310 1315 1320 1325 1330 1335 1340 1345 1350 1355 1360 1365 1370 1375 1380 1385 1390 1395 1400 1405 1410 1415 1420 1425 1430 1435 1440 1445 1450 1455 1460 1465 1470 1475 1480 1485 1490 1495 1500 1505 1510 1515 1520 1525 1530 1535 1540 1545 1550 1555 1560 1565 1570 1575 1580 1585 1590 1595 1600 1605 1610 1615 1620 1625 1630 1635 1640 1645 1650 1655 1660 1665 1670 1675 1680 1685 1690 1695 1700 1705 1710 1715 1720 1725 1730 1735 1740 1745 1750 1755 1760 1765 1770 1775 1780 1785 1790 1795 1800 1805 1810 1815 1820 1825 1830 1835 1840 1845 1850 1855 1860 1865 1870 1875 1880 1885 1890 1895 1900 1905 1910 1915 1920 1925 1930 1935 1940 1945 1950 1955 1960 1965 1970 1975 1980 1985 1990 1995 2000 2005 2010 2015 2020 2025 2030 2035 2040 2045 2050 2055 2060 2065 2070 2075 2080 2085 2090 2095 2100 2105 2110 2115 2120 2125 2130 2135 2140 2145 2150 2155 2160 2165 2170 2175 2180 2185 2190 2195 2200 2205 2210 2215 2220 2225 2230 2235 2240 2245 2250 2255 2260 2265 2270 2275 2280 2285 2290 2295 2300 2305 2310 2315 2320 2325 2330 2335 2340 2345 2350 2355 2360 2365 2370 2375 2380 2385 2390 2395 2400 2405 2410 2415 2420 2425 2430 2435 2440 2445 2450 2455 2460 2465 2470 2475 2480 2485 2490 2495 2500 2505 2510 2515 2520 2525 2530 2535 2540 2545 2550 2555 2560 2565 2570 2575 2580 2585 2590 2595 2600 2605 2610 2615 2620 2625 2630 2635 2640 2645 2650 2655 2660 2665 2670 2675 2680 2685 2690 2695 2700 2705 2710 2715 2720 2725 2730 2735 2740 2745 2750 2755 2760 2765 2770 2775 2780 2785 2790 2795 2800 2805 2810 2815 2820 2825 2830 2835 2840 2845 2850 2855 2860 2865 2870 2875 2880 2885 2890 2895 2900 2905 2910 2915 2920 2925 2930 2935 2940 2945 2950 2955 2960 2965 2970 2975 2980 2985 2990 2995 3000 3005 3010 3015 3020 3025 3030 3035 3040 3045 3050 3055 3060 3065 3070 3075 3080 3085 3090 3095 3100 3105 3110 3115 3120 3125 3130 3135 3140 3145 3150 3155 3160 3165 3170 3175 3180 3185 3190 3195 3200 3205 3210 3215 3220 3225 3230 3235 3240 3245 3250 3255 3260 3265 3270 3275 3280 3285 3290 3295 3300 3305 3310 3315 3320 3325 3330 3335 3340 3345 3350 3355 3360 3365 3370 3375 3380 3385 3390 3395 3400 3405 3410 3415 3420 3425 3430 3435 3440 3445 3450 3455 3460 3465 3470 3475 3480 3485 3490 3495 3500 3505 3510 3515 3520 3525 3530 3535 3540 3545 3550 3555 3560 3565 3570 3575 3580 3585 3590 3595 3600 3605 3610 3615 3620 3625 3630 3635 3640 3645 3650 3655 3660 3665 3670 3675 3680 3685 3690 3695 3700 3705 3710 3715 3720 3725 3730 3735 3740 3745 3750 3755 3760 3765 3770 3775 3780 3785 3790 3795 3800 3805 3810 3815 3820 3825 3830 3835 3840 3845 3850 3855 3860 3865 3870 3875 3880 3885 3890 3895 3900 3905 3910 3915 3920 3925 3930 3935 3940 3945 3950 3955 3960 3965 3970 3975 3980 3985 3990 3995 4000 4005 4010 4015 4020 4025 4030 4035 4040 4045 4050 4055 4060 4065 4070 4075 4080 4085 4090 4095 4100 4105 4110 4115 4120 4125 4130 4135 4140 4145 4150 4155 4160 4165 4170 4175 4180 4185 4190 4195 4200 4205 4210 4215 4220 4225 4230 4235 4240 4245 4250 4255 4260 4265 4270 4275 4280 4285 4290 4295 4300 4305 4310 4315 4320 4325 4330 4335 4340 4345 4350 4355 4360 4365 4370 4375 4380 4385 4390 4395 4400 4405 4410 4415 4420 4425 4430 4435 4440 4445 4450 4455 4460 4465 4470 4475 4480 4485 4490 4495 4500 4505 4510 4515 4520 4525 4530 4535 4540 4545 4550 4555 4560 4565 4570 4575 4580 4585 4590 4595 4600 4605 4610 4615 4620 4625 4630 4635 4640 4645 4650 4655 4660 4665 4670 4675 4680 4685 4690 4695 4700 4705 4710 4715 4720 4725 4730 4735 4740 4745 4750 4755 4760 4765 4770 4775 4780 4785 4790 4795 4800 4805 4810 4815 4820 4825 4830 4835 4840 4845 4850 4855 4860 4865 4870 4875 4880 4885 4890 4895 4900 4905 4910 4915 4920 4925 4930 4935 4940 4945 4950 4955 4960 4965 4970 4975 4980 4985 4990 4995 5000 5005 5010 5015 5020 5025 5030 5035 5040 5045 5050 5055 5060 5065 5070 5075 5080 5085 5090 5095 5100 5105 5110 5115 5120 5125 5130 5135 5140 5145 5150 5155 5160 5165 5170 5175 5180 5185 5190 5195 5200 5205 5210 5215 5220 5225 5230 5235 5240 5245 5250 5255 5260 5265 5270 5275 5280 5285 5290 5295 5300 5305 5310 5315 5320 5325 5330 5335 5340 5345 5350 5355 5360 5365 5370 5375 5380 5385 5390 5395 5400 5405 5410 5415 5420 5425 5430 5435 5440 5445 5450 5455 5460 5465 5470 5475 5480 5485 5490 5495 5500 5505 5510 5515 5520 5525 5530 5535 5540 5545 5550 5555 5560 5565 5570 5575 5580 5585 5590 5595 5600 5605 5610 5615 5620 5625 5630 5635 5640 5645 5650 5655 5660 5665 5670 5675 5680 5685 5690 5695 5700 5705 5710 5715 5720 5725 5730 5735 5740 5745 5750 5755 5760 5765 5770 5775 5780 5785 5790 5795 5800 5805 5810 5815 5820 5825 5830 5835 5840 5845 5850 5855 5860 5865 5870 5875 5880 5885 5890 5895 5900 5905 5910 5915 5920 5925 5930 5935 5940 5945 5950 5955 5960 5965 5970 5975 5980 5985 5990 5995 6000 6005 6010 6015 6020 6025 6030 6035 6040 6045 6050 6055 6060 6065 6070 6075 6080 6085 6090 6095 6100 6105 6110 6115 6120 6125 6130 6135 6140 6145 6150 6155 6160 6165 6170 6175 6180 6185 6190 6195 6200 6205 6210 6215 6220 6225 6230 6235 6240 6245 6250 6255 6260 6265 6270 6275 6280 6285 6290 6295 6300 6305 6310 6315 6320 6325 6330 6335 6340 6345 6350 6355 6360 6365 6370 6375 6380 6385 6390 6395 6400 6405 6410 6415 6420 6425 6430 6435 6440 6445 6450 6455 6460 6465 6470 6475 6480 6485 6490 6495 6500 6505 6510 6515 6520 6525 6530 6535 6540 6545 6550 6555 6560 6565 6570 6575 6580 6585 6590 6595 6600 6605 6610 6615 6620 6625 6630 6635 6640 6645 6650 6655 6660 6665 6670 6675 6680 6685 6690 6695 6700 6705 6710 6715 6720 6725 6730 6735 6740 6745 6750 6755 6760 6765 6770 6775 6780 6785 6790 6795 6800 6805 6810 6815 6820 6825 6830 6835 6840 6845 6850 6855 6860 6865 6870 6875 6880 6885 6890 6895 6900 6905 6910 6915 6920 6925 6930 6935 6940 6945 6950 6955 6960 6965 6970 6975 6980 6985 6990 6995 7000 7005 7010 7015 7020 7025 7030 7035 7040 7045 7050 7055 7060 7065 7070 7075 7080 7085 7090 7095 7100 7105 7110 7115 7120 7125 7130 7135 7140 7145 7150 7155 7160 7165 7170 7175 7180 7185 7190 7195 7200 7205 7210 7215 7220 7225 7230 7235 7240 7245 7250 7255 7260 7265 7270 7275 7280 7285 7290 7295 7300 7305 7310 7315 7320 7325 7330 7335 7340 7345 7350 7355 7360 7365 7370 7375 7380 7385 7390 7395 7400 7405 7410 7415 7420 7425 7430 7435 7440 7445 7450 7455 7460 7465 7470 7475 7480 7485 7490 7495 7500 7505 7510 7515 7520 7525 7530 7535 7540 7545 7550 7555 7560 7565 7570 7575 7580 7585 7590 7595 7600 7605 7610 7615 7620 7625 7630 7635 7640 7645 7650 7655 7660 7665 7670 7675 7680 7685 7690 7695 7700 7705 7710 7715 7720 7725 7730 7735 7740 7745 7750 7755 7760 7765 7770 7775 7780 7785 7790 7795 7800 7805 7810 7815 7820 7825 7830 7835 7840 7845 7850 7855 7860 7865 7870 7875 7880 7885 7890 7895 7900 7905 7910 7915 7920 7925 7930 7935 7940 7945 7950 7955 7960 7965 7970 7975 7980 7985 7990 7995 8000 8005 8010 8015 8020 8025 8030 8035 8040 8045 8050 8055 8060 8065 8070 8075 8080 8085 8090 8095 8100 8105 8110 8115 8120 8125 8130 8135 8140 8145 8150 8155 8160 8165 8170 8175 8180 8185 8190 8195 8200 8205 8210 8215 8220 8225 8230 8235 8240 8245 8250 8255 8260 8265 8270 8275 8280 8285 8290 8295 8300 8305 8310 8315 8320 8325 8330 8335 8340 8345 8350 8355 8360 8365 8370 8375 8380 8385 8390 8395 8400 8405 8410 8415 8420 8425 8430 8435 8440 8445 8450 8455 8460 8465 8470 8475 8480 8485 8490 8495 8500 8505 8510 8515 8520 8525 8530 8535 8540 8545 8550 8555 8560 8565 8570 8575 8580 8585 8590 8595 8600 8605 8610 8615 8620 8625 8630 8635 8640 8645 8650 8655 8660 8665 8670 8675 8680 8685 8690 8695 8700 8705 8710 8715 8720 8725 8730 8735 8740 8745 8750 8755 8760 8765 8770 8775 8780 8785 8790 8795 8800 8805 8810 8815 8820 8825 8830 8835 8840 8845 8850 8855 8860 8865 8870 8875 8880 8885 8890 8895 8900 8905 8910 8915 8920 8925 8930 8935 8940 8945 8950 8955 8960 8965 8970 8975 8980 8985 8990 8995 9000 9005 9010 9015 9020 9025 9030 9035 9040 9045 9050 9055 9060 9065 9070 9075 9080 9085 9090 9095 9100 9105 9110 9115 9120 9125 9130 9135 9140 9145 9150 9155 9160 9165 9170 9175 9180 9185 9190 9195 9200 9205 9210 9215 9220 9225 9230 9235 9240 9245 9250 9255 9260 9265 9270 9275 9280 9285 9290 9295 9300 9305 9310 9315 9320 9325 9330 9335 9340 9345 9350 9355 9360 9365 9370 9375 9380 9385 9390 9395 9400 9405 9410 9415 9420 9425 9430 9435 9440 9445 9450 9455 9460 9465 9470 9475 9480 9485 9490 9495 9500 9505 9510 9515 9520 9525 9530 9535 9540 9545 9550 9555 9560 9565 9570 9575 9580 9585 9590 9595 9600 9605 9610 9615 9620 9625 9630 9635 9640 9645 9650 9655 9660 9665 9670 9675 9680 9685 9690 9695 9700 9705 9710 9715 9720 9725 9730 973

Parser-Prozeduren gemeinsam vorkommt. t und t' sind Zeiger auf die aktuellen Zeiger der jeweiligen Spitze des Status-Stacks. Schließlich sei $s_1, s_2 \dots s_i \dots s_l$ der eine Status-Stack, während $s'_1, s'_2 \dots s'_{l'} \dots s'_{l''}$ den anderen Status-Stack beschreibt. $v_1, v_2 \dots v_l \dots v_i$ ist der eine Werte-Stack und $v'_1, v'_2 \dots v'_{l'} \dots v'_{l''}$ der andere. $v'_1 \dots v'_{l''}$ der andere. $v'_2 \dots v'_{l'} \dots v'_{l''}$ der andere.

Wenn folgende Bedingungen erfüllt sind, können die beiden Parser-Prozeduren zusammengeführt werden:

1. Die zugehörige **#endif**-Anweisung wurde verarbeitet.
2. $f = f'$
3. $s_{l+1} = s'_{l'+1}$
4. $t = f + 1$
5. $t' = f' + 1$

Ein informeller Beweis für diese Angaben wäre: Punkt 1 ist per Definition erforderlich.

Punkt 2 und Punkt 3 stellen sicher, daß die beiden Parser-Prozeduren vom Anfang des Stacks bis zur Stelle s_i und $s'_{l'}$ identisch sind. Dazu erinnert sei, daß f und f' immer auf die höchste Position auf dem Stack zeigen, bis zu der die beiden Parser-Prozeduren gleich sind. Diese Invariante wird während der gesamten Zeitdauer einer aufgespaltenen Parser-Prozedur beibehalten, indem f zum Zeitpunkt der Aufspaltung auf den Wert t gesetzt wird. Es kann sein, daß f sofort nach der Durchführung einer Reduktion in beiden Parser-Prozeduren auf $t-1$ gesetzt werden muß. Demzufolge sind beide Stacks in den Positionen $1 \dots f+1$ identisch. Da die beiden Parser-Prozeduren daraufhin einen gemeinsamen Eingabestrom verarbeiten, weisen sie im weiteren Verlauf dieselben Zustände auf.

Punkt 4 und 5 garantieren, daß oberhalb des gemeinsamen Anteils ein Wert auf dem Werte-Stack steht. Da eine Zusammenführung durch eine Verschmelzung dieser beiden Werte zu einem gemeinsamen Wert erzeugt wird, gehen keine Informationen verloren.

Die obigen Bedingungen sind hinreichend. Es wird darüber hinaus angenommen, daß sie notwendig sind.

Die Verarbeitungsschritte lauten:

1. Anlegen einer ersten Parser-Prozedur und deren Status auf BlockedOnInput setzen.
2. Solange weitere Parser-Prozeduren vorhanden sind:

- (a) Wenn keine Parser-Prozedur mehr aktiv ist (d.h. alle Parser-Prozeduren haben das aktuelle Token geprüft), wird das nächste Token eingelesen und alle Parser-Prozeduren, die sich im Status BlockOnInput befinden, werden auf den Status Active gesetzt.
- (b) Solange das aktuelle Token eine Präprozessor-Anweisung ist:

- i. Wenn das aktuelle Token eine **#if**,

#ifdef- oder **#ifndef**-Anweisung ist, wird die aktuelle Spitze des Status-Stacks für jede aktive Parser-Prozedur aufgezeichnet, indem ein Zeiger auf die aktuelle Spitze des Status-Stacks auf den ForkPtrStack gelegt wird. Damit kann später festgestellt werden, wann sich die beiden Parser-Prozeduren wieder zusammenführen lassen. Die Parser-Prozeduren werden anschließend kopiert und die kopierten Parser-Prozeduren werden in den Zustand Sleeping gesetzt.

ii. Wenn das aktuelle Token eine **#else**-Anweisung ist, wird die Operation ForkExchange aufgerufen. Aktive Parser-Prozeduren werden ausgesetzt, und die Parser-Prozeduren, die bei dieser **#else**-Anweisung ausgesetzt waren, werden aktiviert.

iii. Wenn es sich beim aktuellen Token um eine **#endif**-Anweisung handelt, werden alle Parser-Prozeduren aktiviert, die bei dieser **#endif**-Anweisung ausgesetzt waren. Anschließend wird versucht, die beiden zuvor getrennten Parser-Prozeduren wieder miteinander zu verbinden. Die beiden Parser-Prozeduren können miteinander verbunden werden, wenn die Kriterien für eine sichere Zusammenführung erfüllt sind. Falls die beiden Parser-Prozeduren nicht zusammengeführt werden können, da sie in unterschiedlichen Konfigurationen vorliegen, vermerkt die Parser-Verwaltung, daß eine Zusammenführung offen ist. Falls die Zusammenführung stattfindet, wird ein Wert vom ForkPtrStack entfernt und eine der beiden Parser-Prozeduren wird beendet.

iv. Nächstes Token einlesen.

(c) Abschließend wählt die Parser-Verwaltung die aktive Parser-Prozedur mit dem höchsten Status-Stack aus und führt die ausgewählte Parser-Prozedur durch einen Iterationszyklus des LALR-Treibers.

Das Ziel in Schritt 2c ist die Zusammenführung

zweier Parser-Prozeduren, um die durch eine bedingte Kompliierungsanweisung verursachten Unterschiede zwischen zwei Parser-Prozeduren so gering wie möglich zu halten. Da sich eine Parser-Prozedur in einem Zustand befinden kann, in dem sie nicht mehr aufeinanderfolgend reduziert werden kann, ohne daß zwischendurch ein Shift erfolgt, muß man darauf achten, daß die Reduktionsoperationen in den Parser-Prozeduren in einer heuristisch festzustellenden Reihenfolge abgesetzt werden, damit die Zusammenführung so früh wie möglich vorgenommen werden kann. Diese heuristische Methode, die für die Beispielausführung mit der C-Grammatik sehr gut funktioniert, wählt die

Parser-Prozedur mit dem höchsten Status-Stack aus. Wenn die beiden Status-Stacks gleich hoch sind, wird die Parser-Prozedur ausgewählt, deren Bearbeitung am weitesten zurückliegt. Diese Strategie sorgt dafür, daß die Status-Stacks der ausgewählten Parser-Prozeduren näher an das Niveau der anderen aktiven Parser-Prozeduren heranrücken und somit keine Planung des Reduktionsablaufs erforderlich ist, wie dies zum Ermitteln für eine optimale Reduktionsstrategie erforderlich ist.

Der modifizierte LALR-Treiber

Diese Prozedur wird von der Parser-Verwaltung aufgerufen, um die ausgewählte Parser-Prozedur in einem Shift-Reduktions-Schritt zu verarbeiten. Der LALR-Treiber wurde von Aho, Sethi und Ullman, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, Reading, MA, 1986, übernommen. Der modifizierte Treiber unterscheidet sich von diesem dadurch, daß der Status-Stack nur Zustände enthält und er nur auf die aktuell ausgewählte Parser-Prozedur angewendet wird. Der Algorithmus lautet:

1. Wenn action[CurState,token] = shift s, s auf den Status-Stack legen und den Status von Active in BlockedOnInput umwandeln.
2. Wenn action[CurState,token] = reduce A -> B, dann reduce A -> B. Die Reduktion wurde geringfügig modifiziert, um mehrere Parser-Prozeduren miteinander zu verbinden:
 - (a) Bei jedem Grammatiksymbol im String β wird ein Element vom Status-Stack genommen.
 - (b) Wenn eine Parser-Zusammenführung ansteht und die Höhe des Status-Stack unterhalb des Wertes liegt, der an der Spitze von ForkPtrStack liegt, wird der Zeiger angepaßt, der an der Spitze von ForkPtrStack liegt, um auf den aktuellen Status-Stack zu verweisen.
 - (c) Wenn eine Parser-Zusammenführung ansteht und die beiden Parser-Prozeduren die Kriterien für eine sichere Zusammenführung erfüllen, werden sie zusammengeführt. Sobald die Zusammenführung abgeschlossen ist, wird der oberste Wert von ForkPtrStack geholt.
 - (d) CurState auf Goto[CurState,A] setzen und CurState auf den Status-Stack legen.
3. Wenn action[CurState,token] = accept, ist die Parser-Prozedur abgeschlossen.
4. Wenn action[CurState,token] = error, wird die Parser-Prozedur verworfen.

Die Darstellung echter C-Programme

Mit dem Parson von C-Programmen unter Erhalt der Präprozessor-Informationen hat man nur die Hälfte des Problems gelöst. Die andere Hälfte der Aufgabe ist die Darstellung dieser Informationen in einem abstrakten Syntaxbaum. Der schwierigste Punkt betrifft die Probleme durch die bedingte

Kompilierung. Wie in FIG 1 gezeigt, ergeben sich durch die bedingte Kompilierung mehrere Versionen eines Programms aus einem Quelltext. Ein einfacher Ansatz zur Darstellung dieser verschiedenen Versionen wäre die Einführung eines Knotens im abstrakten Syntaxbaum, der den bedingt komplizierten Code darstellt. Dieser Knoten hätte zwei weitere Unterbäume als Nachfolger, jeweils einen Unterbaum für jede Alternative einer **#ifdef**-Anweisung. Diese Unterbäume würden durch die aufgespaltenen Parser-Prozeduren erzeugt und durch eine ParseJoin-Operation würde der Knoten über einen Verbindungsknoten **Join** mit seinen Nachfolgern verbunden. Dies ist jedoch nicht so einfach, wie es scheint. Es ergeben sich folgende Probleme:

- Da ein LALR-Parser verwendet wird, können Reduktionen und die damit verbundenen Aktionen nur Bottom-Up (nach Analyse aller Syntax-Details) durchgeführt werden, so daß sich eine Parser-Prozedur mit einer rechtsorientierten Ableitung in umgekehrter Reihenfolge ergibt. Noch problematischer wird die Angelegenheit dadurch, daß die Implementierung »links-rekursive« Grammatikregeln begünstigt. Zum Beispiel:

```
list : item
      list ; item
      ;
```

Dies wird zu einem Problem, wenn wir versuchen, Bäume mit gemeinsamen Vorfahren miteinander zu verknüpfen. Wenn zum Beispiel in einer Anweisungsfolge mit zehn Elementen das letzte Element bedingt kompiliert wird, enthalten die beiden Syntaxbäume Kopien des Unterbaums der ersten neun Anweisungen der Folge. Diese Wiederholung ist natürlich sehr verschwendärisch.

- In gleicher Weise ergibt sich beliebig viel Code, wenn die Schritte für eine **#endif**-Anweisung in den Nachfolgern eines Verbindungsknotens dupliziert werden. Dieser Code ist nötig, um die syntaktischen Einheiten zu vervollständigen, die innerhalb von **#ifdef** anfangen. Anweisung S4 in Abbildung 1 verdeutlicht diesen Fall. Die Kopie dieses Codes würde den Teil, der bedingt kompiliert wird, nur sehr ungenau darstellen.

Diese und viele Probleme ergeben sich wegen der Wiederholungen, die durch eine baumartige Darstellung erforderlich werden. Dies läßt sich durch diese Erfindung lösen, indem man eine Darstellung wählt, die auf Graphen basiert, die im weiteren Verlauf als abstrakte Syntaxgraphen (ASG) bezeichnet werden. Anstelle der Verdopplung gemeinsamer Unterbäume, verweist man von mehreren Stellen aus auf eine gemeinsame Posi-

tion. FIG 2 zeigt den abstrakten Syntaxgraphen für das Programm von FIG 1. Dabei ist zu beachten, daß der Verbindungsknoten **Join** eingefügt wurde, um die Stelle zu markieren, an der die Parser-Prozeduren wieder zusammenlaufen. Die Nachfolger dieses Knotens repräsentieren die beiden Hälften der **#ifdef**-Anweisung. Die Knoten der Anweisungsfolge direkt über S1 und die Anweisung S4 kommen in beiden Nachfolgern vor. Obwohl eine Darstellung in Form eines Graphen etwas schwieriger zu durchwandern ist, gibt sie die Struktur des ursprünglichen Programms exakt wieder.

Die Erzeugung dieser Darstellung erfordert die Behandlung von drei Problemen:

1. Die Prozedur muß einen Knoten für die Verbindung aufbauen und sicherstellen, daß alle Knoten, die Code aus der bedingten Kompilierung darstellen, Nachfolger dieses Knotens sind.
2. Es müssen Verweise auf die Knoten erzeugt werden, in denen der gemeinsame Code dargestellt wird, bevor man bei einer **#ifdef**-Anweisung wieder zu den ursprünglichen Knoten zurückkehrt.
3. Dasselbe ist für Knoten zu bewerkstelligen, die den Code nach der **#endif**-Anweisung darstellen.

Diese Probleme werden mit Hilfe eines Werte-Stacks gelöst. Jeder Eintrag auf dem Werte-Stack ist ein Zeiger auf einen Knoten im abstrakten Syntaxgraphen. Findet während des Parsens eine Reduktion statt, wird die entsprechende Aktionsroutine ausgeführt. Diese Routine nimmt eine Reihe von Einträgen vom Werte-Stack und erzeugt einen neuen Knoten, der die vom Stack heruntergeholt Zeiger enthält. Anschließend wird ein Zeiger auf diesen Knoten auf den Werte-Stack gelegt. Diese Methode wird von LALR-Parsern standardmäßig unterstützt.

Das Anlegen des Verbindungsknotens **Join** an der richtigen Stelle liegt außerhalb der Kriterien für die Zusammenführung von Parser-Prozeduren. Der Verbindungsknoten **Join** wird in einer Rücksprung-Routine erzeugt, die bei der Zusammenführung zweier Parser-Prozeduren aufgerufen wird. Die obersten Einträge des Werte-Stacks der beiden zusammengeführten Parser-Prozeduren verweisen auf Untergraphen, die den Code darstellen, der im **#if**- bzw. **#else**-Abschnitt liegt. Alles, was unterhalb dieser Einträge liegt, lag bereits vor, als die Parser-Prozeduren aufgespalten wurden. Da sich die Parser-Prozeduren an dieser Stelle in identischen Zuständen befinden, müssen die Knotentypen am Ausgangspunkt dieser Unterbäume gleich sein. Die Knoten können durch einen einzigen Verbindungsknoten **Join** ersetzt werden, der auf die Unterbäume verweist.

Die Verbindung der Knoten durch Verweise zum Darstellen der gemeinsamen Code-Abschnitte

vor einer **#ifdef**-Anweisung liegt ebenfalls außerhalb der Parser-Prozedur. Da die Einträge auf den Werte-Stacks nur Verweise auf Knoten im abstrakten Syntaxgraphen darstellen und selbst keine Knoten sind, werden durch das Kopieren des Werte-Stacks während der Aufspaltung einer Parser-Prozedur keine Kopien von Knoten angelegt. Es werden lediglich Kopien der Knotenverweise erzeugt. Wenn also die Reduktionen in den abgespaltenen Parser-Prozeduren stattfinden, enthalten die erstellten Knoten nur kopierte Verweise, und keine Kopien der Knoten selbst.

Derselbe Vorgang für die Knoten hinter der **#endif**-Anweisung gestaltet sich wesentlich schwieriger. Die vorliegende Parser-Strategie unterstützt dies nicht. Für die Realisierung benötigt man eine Kommunikationsmöglichkeit zwischen den abgespaltenen Parser-Prozeduren, damit Informationen über die Zustände ausgetauscht werden können, in denen sich die Parser-Prozeduren befinden. Momentan ist dieses Problem noch nicht gelöst. Letztlich wird ein Graph erzeugt, indem die **#endif**-Anweisung auf den Verbindungsknoten **Join** des abstrakten Syntaxgraphen kopiert wird. Semantisch gesehen enthält der erzeugte Graph alle Informationen des ursprünglichen Programms: es ist jedoch eine platzsparendere Darstellung möglich.

Ein weiteres Problem ergibt sich bei der Berücksichtigung der Präprozessor-Informationen von Makrodefinitionen in der Darstellung von C-Programmen. Es wäre wünschenswert, wenn man beide Formen eines Makros, vor und nach der Ersetzung des Aufrufs, im abstrakten Syntaxgraphen darstellen könnte. Dies ist aber nicht immer möglich, da das Ergebnis einer Makro-Ersetzung nicht notwendigerweise syntaktisch abgeschlossen sein muß, d.h. es muß sich nicht unbedingt ein Unterbaum des abstrakten Syntaxgraphen ergeben. Wir haben einen Kompromiß entwickelt, um dieses Problem zu lösen. Der Anwender hat zwei Alternativen. In der ersten Alternative werden die Abschnitte des abstrakten Syntaxgraphen, die sich aus der Ersetzung eines Makros ergeben, markiert, es wird aber nicht versucht, die Aufruf ohne Ersetzung darzustellen. Wenn der Anwender jedoch garantieren kann, daß keine syntaktisch unvollständigen Makros vorkommen, wird derselbe Mechanismus verwendet, um die ersetzen und die nicht ersetzen Formen von Makros darzustellen. Ein eigener Parser erzeugt mit Hilfe der cpp-Grammatik einen Unterbaum für die nicht ersetze Form. Der Hauptparser wird aufgespalten und mit dem ersetzen Text versorgt. Anschließend wird mit ParseJoin eine Zusammenführung signalisiert, in der ein Verbindungsknoten **Join** mit nur einem Nachfolger angelegt wird. Anschließend wird der Unterbaum mit der nicht ersetzen Form des Makros eingebunden. Der Verbindungsknoten **Join** wird mit Attributen

versehen, die anzeigen, daß es sich um das Ergebnis eines Makro-Aufrufs handelt, und nicht um eine bedingte Kompilierung.

Die übrigen Eigenschaften des C-Präprozessors, die noch dargestellt werden müssen, wie **#include**, **#define**, Kommentare, usw., werden durch Attribute der Knoten des Graphs behandelt. Da diese Eigenschaften an einer beliebigen Stelle im Programm auftreten können, ist es nicht möglich, sie in Form von Knoten innerhalb des abstrakten Syntaxgraphen darzustellen. Aus Platzgründen taucht im Beispiel keine Übernahme einer Datei mit Hilfe der **#include**-Anweisung auf. Da Include-Dateien üblicherweise für Deklarationen verwendet werden, wird lediglich die Information der Symboltabelle aus der Include-Datei übernommen.

Ansprüche

1. Verfahren für die automatische Syntaxanalyse (Parsen) des Textes von Computer-Programmen in Kompilierern und entsprechenden Einrichtungen, wobei der Text bedingte Kompilierungsanweisungen enthält, aus denen sich mehrere Versionen eines Programms aus einem einzigen Text ableiten,

gekennzeichnet durch

- Multiplexen und Synchronisieren mehrerer gleichzeitig aktiver Parser-Prozeduren, die einen gemeinsamen Eingabestrom des Textes verarbeiten;
- Abspalten einer neuen Parser-Prozedur beim Auftreten von bedingten Kompilierungsanweisungen im Eingabestrom, die anzeigen, daß direkt nachfolgende Eingaben eine Variante aus einer Vielzahl von möglichen Programmvarianten darstellen, als ParseFork-Operation;
- Zusammenführen von zwei zuvor aufgespalteten Parser-Prozeduren, falls alle Kriterien für eine Zusammenführung erfüllt sind, als ParseJoin-Operation, sowie
- Festlegen der Reihenfolge der Shift-Reduktions-Operationen in einem LR-Treiber für die miteinander synchronisierten Parser-Prozeduren.

2. Verfahren nach Anspruch 1, **gekennzeichnet durch** den Austausch einer Parser-Prozedur, die einem Verzweigungspfad der bedingten Kompilierungsanweisung zugeordnet ist, durch die Parser-Prozedur des anderen Verzweigungspfads der bedingten Kompilierungsanweisung als ParseExchange-Operation.

3. Verfahren nach Anspruch 2,

dadurch gekennzeichnet, daß nach Auftreten einer bedingten Kompilierungsanweisung im Eingabestrom eine neue Parser-Prozedur abgespalten wird und eine Kopie des aktuellen Parser-Status gespeichert wird, damit der Eingabestrom von mehr als einer Parser-Prozedur verarbeitet und später festgestellt werden kann, ob sich die beiden Parser-Prozeduren in einem Zustand befinden, in dem sie zusammengeführt werden können.

4. Verfahren nach Anspruch 3,

dadurch gekennzeichnet, daß zwei zuvor aufgespaltene Parser-Prozeduren Reduktionen in heuristisch festgelegter Reihenfolge unterzogen werden, damit die beiden Parser-Prozeduren mit der kleinstmöglichen Anzahl an Shift-Reduktions-Operationen miteinander verbunden werden können.

5. Verfahren nach Anspruch 4,

dadurch gekennzeichnet, daß jede Parser-Prozedur einen Status-Stack und einen Werte-Stack besitzt, die miteinander synchronisiert sind und daß, falls die zugehörige **#endif**-Anweisung angetroffen wird und beide aufgespalteten Parser-Prozeduren identische Zustände aufweisen sowie im Werte-Stack gespeicherte Werte bei einer Zusammenführung nicht verloren gehen, alle Zusammenführungskriterien erfüllt sind und eine Zusammenführung vollzogen wird, ohne daß Informationen verlorengehen.

6. Verfahren nach einem der Ansprüche 1 bis 5, **gekennzeichnet durch** folgende Verfahrensschritte:

i. Solange verarbeitbare Teile des Textes als Token zur Eingabe verfügbar sind: Auswahl des nächsten Tokens und Wirmschalten aller Parser-Prozeduren, die von der Eingabe ausgeschlossen waren, so daß das neue Token von diesen Parser-Prozeduren verarbeitet werden kann.

ii. Solange eine Parser-Prozedur für die Bearbeitung der aktuellen Eingabe verfügbar ist, Auswahl der nächsten verfügbaren Parser-Prozedur für die Verarbeitung des Tokens mit folgenden Operationen:

A. Wenn action[CurState,token] = shift s, s auf den Status-Stack legen und diese Parser-Prozedur solange von der Verarbeitung ausschließen, bis alle anderen Parser-Prozeduren das aktuelle Token verarbeitet haben.

B. Wenn action[CurState,token] = reduction A → B, dann reduction A → B durchführen; in dem

1. bei jedem Grammatiksymbol im String β ein Element vom Status-Stack genommen wird,

2. bei einer anstehenden Parser-Zusammenführung und Erfüllung der Kriterien für eine Zusammenführung der beiden betroffenen Parser-Prozeduren die beiden Parser-Prozeduren zusammengeführt werden und

3. CurState auf Goto[CurState,A] gesetzt und CurState auf den Stack gelegt wird.

C. Wenn action[CurState,token] = accept, ist die Parser-Prozedur abgeschlossen.

7. Verfahren nach Anspruch 6, **gekennzeichnet durch** folgende Operation: Wenn action[CurState,token] = error, wird die Parser-Prozedur verworfen.

8. Verfahren nach einem der Ansprüche 1 bis 7, **dadurch gekennzeichnet**, daß unter der Voraussetzung, daß p und p' zwei aufgespaltene Parser-Prozeduren sind, f und f' als Zeiger auf die oberste Position des Status-Stacks zeigen, der für beide Parser-Prozeduren zum Zeitpunkt der Spaltung gemeinsam gilt, t und t' als laufende Zeiger auf die jeweilige Spitze der Status-Stacks beider Parser-Prozeduren zeigen, $s_1, s_2 \dots s_i \dots s_t$ den einen Status-Stack und $s'_1, s'_2 \dots s'_{i'} \dots s'_{t'}$ den anderen Status-Stack beschreiben die Bedingungen für die Zusammenführung der beiden Parser-Prozeduren erfüllt sind, wenn gilt:

1. Die zugehörige **#endif**-Anweisung wurde verarbeitet.
2. $f = f'$
3. $s_{i+1} = s'_{i'+1}$
4. $t = f + 1$
5. $t' = f' + 1$

9. Verfahren nach einem der Ansprüche 1 bis 8, **dadurch gekennzeichnet**, daß die **Parse-Fork**-Operation durch die Kompilierungsanweisungen **#ifdef**, **#ifndef** oder **#if** als Token im Eingabestrom ausgelöst wird, wobei der augenblickliche Parser-Zustand einschließlich des Werte- und Status-Stack der Parser-Prozedur kopiert wird und der Eingabezeiger von allen Parser-Prozeduren gemeinsam verwendet wird, und daß die **ParseExchange**-Operation durch die bedingten Kompilierungsanweisungen **#else** oder **#elif** als Token im Eingabestrom ausgelöst wird, wobei die augenblicklich ausgeführte Parser-Prozedur ausgesetzt wird, während die augenblicklich ausgesetzten Parser-Prozeduren aktiviert und für die Aufnahme von Token vorbereitet werden.

10. Verfahren nach Anspruch 9, **dadurch gekennzeichnet**, daß jede Operation mit einer anwendungsspezifischen Rücksprung-Routine verbunden ist, die bei jeder Operation aufgerufen wird.

11. Verfahren nach Anspruch 9 und 10, **dadurch gekennzeichnet**, daß die Operationen **ParseFork**, **ParseJoin** und **ParseExchange** vom LR-Treiber ausgeführt werden und daß bei der Ausführung einer **ParseJoin**-Operation und Vorliegen zweier wegen nicht identischer Zustände augenblicklich nicht zusammenführbarer Parser-Prozeduren die Zusammenführung aufgeschoben wird und mehrere Parser-Prozeduren aktiv bleiben, bis die Parser-Prozeduren zusammengeführt werden können, wobei die Rücksprung-Routine solange nicht aufgerufen wird, bis die Zusammenführung erfolgt.

12. Verfahren nach einem der Ansprüche 1 bis 11, **dadurch gekennzeichnet**, daß eine Parser-Verwaltung den Eingabestrom des Programms multiplext und bestimmt, welche Parser-Prozedur durch den LR-Treiber laufen soll.

13. Verfahren nach Anspruch 12, **dadurch gekennzeichnet**, daß der Zustand der einzelnen Parser-Prozeduren in einer Variablen in der Parser-Verwaltung festgehalten wird.

14. Verfahren nach Anspruch 13, **dadurch gekennzeichnet**, daß folgende Zustände auftreten können:

- **Active**: Die Parser-Prozedur darf das aktuelle Token aus dem Eingabestrom entgegennehmen.
- **BlockedOnInput**: Das aktuelle Token wurde von der Parser-Prozedur bereits erkannt, und es können keine weiteren Operationen durchgeführt werden können, da die anderen aktiven Parser-Prozeduren bereits darauf warten, das aktuelle Token zu verarbeiten.
- **Sleeping**: Eine Parser-Prozedur wurde ausgesetzt, da sie nichts mit dem aktuellen Eingabestrom zu tun hat, weil eine Verzweigung einer bedingten Kompilierungsanweisung verarbeitet wird und die

andere Verzweigung auf die entsprechende #else- oder #endif-Anweisung wartet.

5

10

15

20

25

30

35

40

45

50

55

10

FIG 1

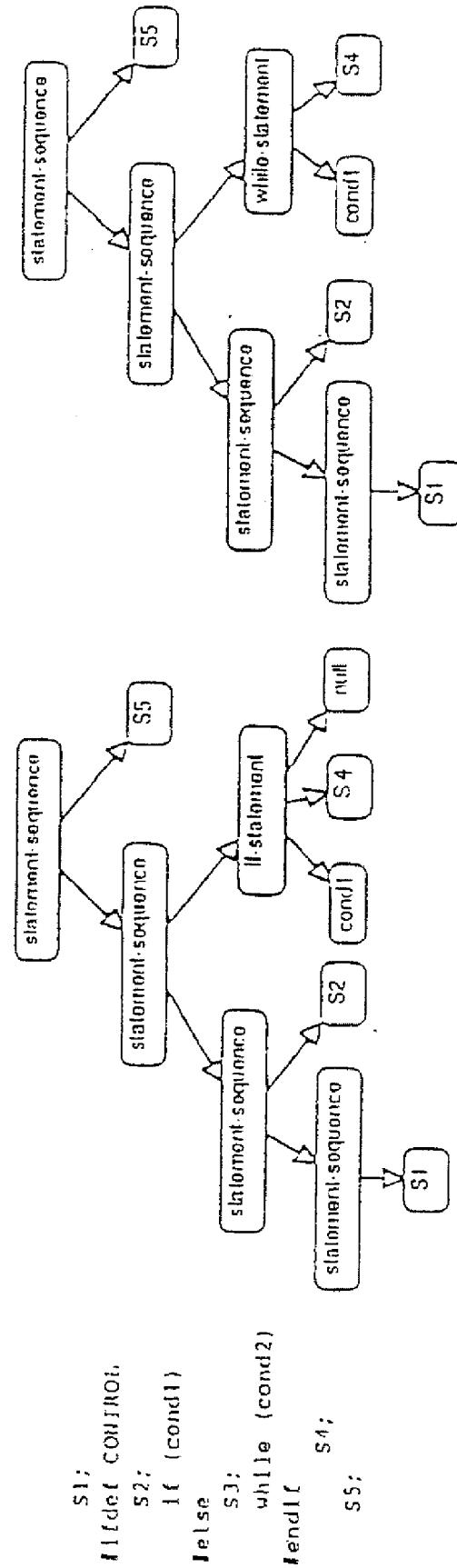
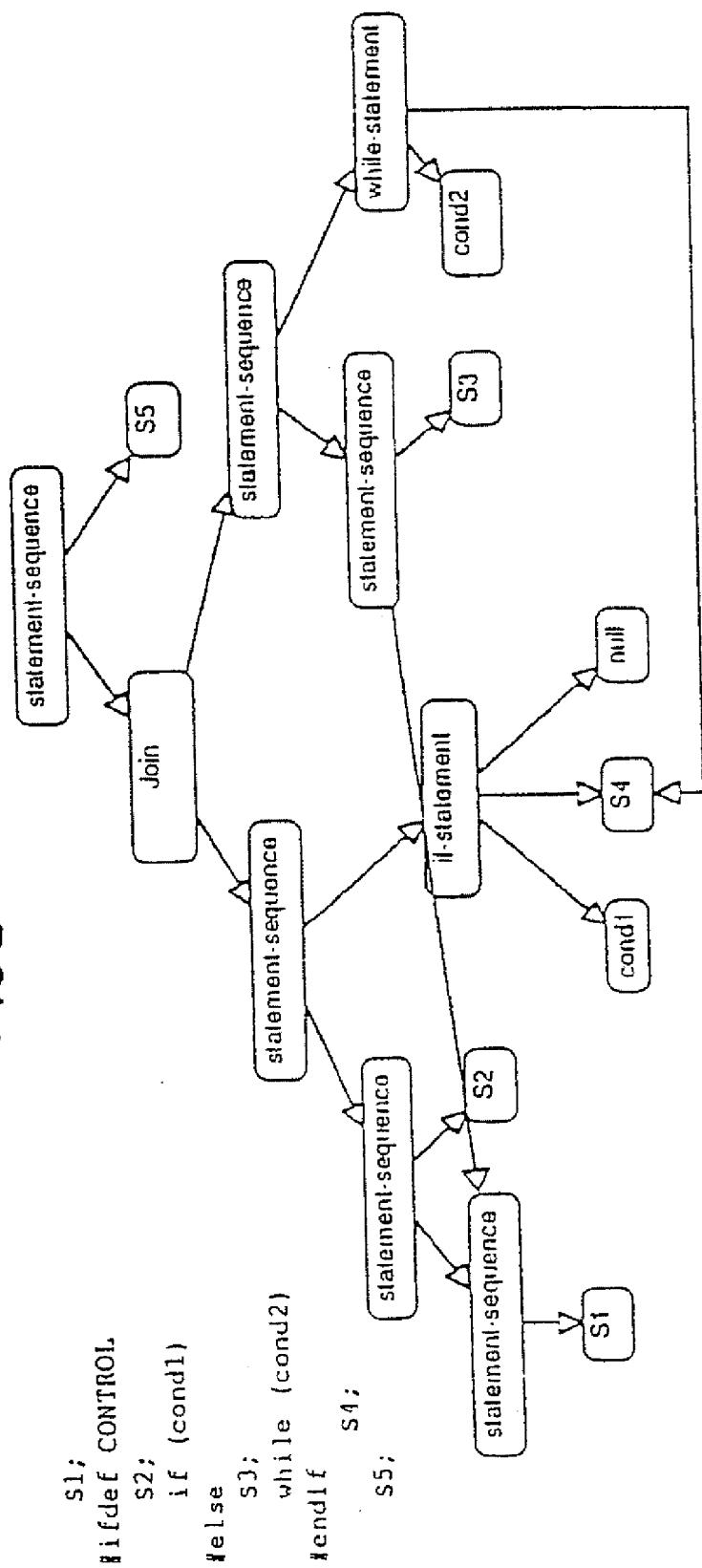


FIG 2





EUROPÄISCHE PATENTANMELDUNG

(21) Anmeldenummer: 90124244.6

(51) Int. Cl. 5: **G06F 9/45**

(22) Anmeldetag: 14.12.90

(30) Priorität: 15.12.89 US 451493

Otto-Hahn-Ring 6
W-8000 München 83(DE)

(43) Veröffentlichungstag der Anmeldung:
19.06.91 Patentblatt 91/25

(72) Erfinder: Platoff, Michael, A.
2151 Ash Court
Monmouth Junction, NJ 08852(US)
Erfinder: Wagner, Michael, E.
21 Crest Avenue
Trenton, NJ 08638(US)

(54) Benannte Vertragsstaaten:
AT BE CH DE ES FR GB IT LI NL SE

(74) Vertreter: Fuchs, Franz-Josef, Dr.-Ing. et al
Postfach 22 13 17
W-8000 München 22 (DE)

(86) Veröffentlichungstag des später veröffentlichten
Recherchenberichts: 23.06.93 Patentblatt 93/25

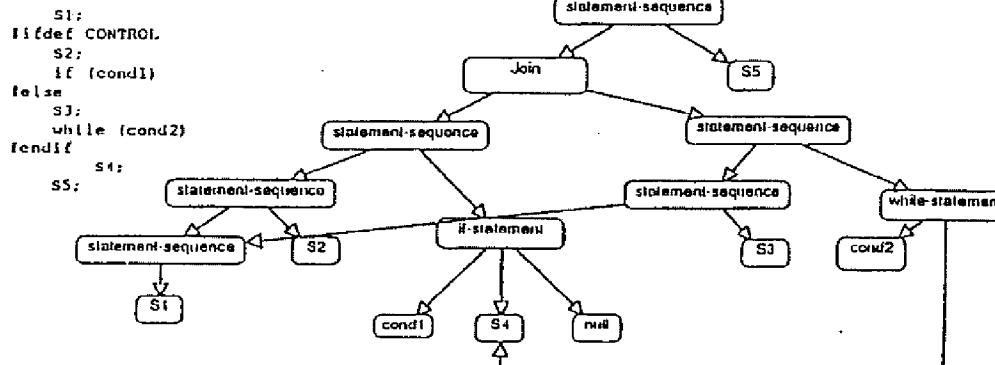
(71) Anmelder: Siemens Nixdorf
Informationssysteme AG

(52) Verfahren für die automatische Syntaxanalyse (Parseen) des Textes von Computer-Programmen in Kompilierern.

(57) Zur Ermittlung und Darstellung der Struktur von Programmen, die wie in C Präprozessor-Anweisungen und Erweiterungen gegenüber der abstrakten Syntaxbaum-Darstellung aufweisen, wird ein modifizierter Parser-Generator mit zugehöriger Parser-Verwaltung verwendet, wobei mehrere Parser-Prozeduren denselben Eingabestrom vom Token verarbeiten und zusätzliche Operationen zum Abspalten neuer Parserprozeduren und zum Zusammenführen von aufgespaltenen Parser-Prozeduren sowie zum Aus-

tausch einer Parser-Prozedur für den einen Zweig einer bedingten Kompilierungsanweisung durch die Parser-Prozeduren für den anderen Verzweigungspfad beim Auftreten von vorgegebenen bedingten Kompilierungsanweisungen ausgeführt werden. Welche der Parser-Prozeduren jeweils durch den Parsergenerator laufen soll, bestimmt die Parserverwaltung, wobei der jeweilige Status der einzelnen Parser-Prozeduren in der Parserverwaltung festgehalten wird.

FIG2





Europäisches
Patentamt

EUROPÄISCHER TEILRECHERCHENBERICHT Nummer der Anmeldung
der nach Regel 45 des Europäischen Patent-
übereinkommens für das weitere Verfahren als
europäischer Recherchenbericht gilt

EP 90 12 4244
Seite 1

EINSCHLÄGIGE DOKUMENTE			
Kategorie	Kennzeichnung des Dokuments mit Angabe, soweit erforderlich der maßgeblichen Teile	Betrieft Anspruch	KLASSIFIKATION DER ANMELDUNG (Int. Cl. 5)
X	ACM TRANSACTIONS ON PROGRAMMING LANGUAGES AND SYSTEMS Bd. 2, Nr. 3, Juli 1980, US Seiten 290 - 306 HANAN SAMET: 'A Coroutine Approach to Parsing'	1-3, 9	G06F9/45
A	* Zusammenfassung * * Seite 290, Zeile 1 - Zeile 11 * * Seite 291, Zeile 12 - Seite 298, Zeile 36; Abbildungen 1-6; Tabelle I * * Seite 299, Zeile 38 - Seite 300, Zeile 2 * ---	14	
A	EP-A-0 343 883 (HEWLETT-PACKARD COMPANY) 29. November 1989 * Zusammenfassung; Abbildung 1 * ---	1	
A	PATENT ABSTRACTS OF JAPAN vol. 9, no. 214 (P-384) 31. August 1985 & JP-A-60 074 039 (FUJITSU K.K.) 26. April 1985 * Zusammenfassung * ---	1	
		-/--	RECHERCHIERTE SACHGEBIETE (Int. Cl. 5) G06F
UNVOLLSTÄNDIGE RECHERCHE			
<p>Nach Auffassung der Recherchenabteilung entspricht die vorliegende europäische Patentanmeldung den Vorschriften des Europäischen Patentübereinkommens so wenig, daß es nicht möglich ist, auf der Grundlage einiger Patentansprüche sinnvolle Ermittlungen über den Stand der Technik durchzuführen.</p> <p>Vollständig recherchierte Patentansprüche: 1-5, 9-14 Unvollständig recherchierte Patentansprüche: 6-8 Nicht recherchierte Patentansprüche: Grund für die Beschränkung der Recherche:</p> <p>Programm für Datenverarbeitungsanlagen (Art. 52(2)C)</p>			
Recherchewort: DEN HAAG	Abschlußdatum der Recherche: 26 APRIL 1993	Präfer	WILTINK J.G.
KATEGORIE DER GENANNTEN DOKUMENTEN		<p>T : der Erfindung zugrunde liegende Theorien oder Grundsätze E : älteres Patentdokument, das jedoch erst am oder nach dem Anmeldedatum veröffentlicht worden ist D : in der Anmeldung angeführtes Dokument L : aus andern Gründen angeführtes Dokument & : Mitglied der gleichen Patentfamilie, übereinstimmendes Dokument</p>	
<p>EPO FORM 150103 (P-94-099)</p>			



Europäisches
Patentamt

EUROPÄISCHER TEILRECHERCHENBERICHT

Nummer der Anmeldung

EP 90 12 4244
Seite 2

EINSCHLÄGIGE DOKUMENTE			KLASSIFIKATION DER ANMELDUNG (Int. Cl. 5)
Kategorie	Kennzeichnung des Dokuments mit Angabe, soweit erforderlich der maßgeblichen Teile	Betrift Anspruch	
A	IBM TECHNICAL DISCLOSURE BULLETIN. Bd. 13, Nr. 10, März 1971, NEW YORK US Seite 2903 T.L. OLIVER: 'Bilingual mapping macro' * das ganze Dokument *	1	RECHERCHIERTE SACHGEBiete (Int. Cl. 5)